Checkers
Testing Document

Group Members: Tyr Zong, Vincent Werkle, Chris Bakopoulos, Josh Romley
Faculty Professor: Professor Alex Palmatier
Project Stakeholders: Widchard Faustin, Saugat Sthapit

1. Introduction

The purpose of this document is to describe the testing that will be done to evaluate the functionality and performance of our Checker's software. This document will work in concert with the requirements document to ensure that we fulfill the requirements set out for us in that document. Our Checker's software is a web app which allows users to play checkers together through their web browser by connecting to our web server.

1.1 Definitions, Acronyms and Abbreviations

Refer to section 5, the Appendix and Glossary for these.

1.2 References

This document may reference previous documents given, the design and requirements documents. Please refer to those if there are any questions about the referred section of the document.

2. Environment

The following tables describe our testing environments.

2.1 Frontend Environments:

Machine Name	Linux VM
OS Name and Version	Ubuntu 16.04.5 LTS 8GB RAM; 256GB HDD
Tester Name	Chris Bakopoulos
Logs	
Web Browsers used	Google Chrome Ver. 74 Firefox Ver. 65

Machine Name	Windows Laptop
OS Name and Version	Windows 10 Ver. 1703 8GB RAM; 256GB HDD
Tester Name	Chris Bakopoulos
Logs	
Web Browsers used	Google Chrome Ver. 74 Firefox Ver. 65 Internet Explorer Ver. 11

2.2 Backend Environments

Machine Name	AWS Instance
OS Name and Version	OpenSUSE 15.0 8GB RAM; 256GB HDD
Tester Name	Chris Bakopoulos
Logs	
Javascript Interpreter	Node.js 11.15.2 LTS

3. Setup Information and Prerequisites

Prior to running this program the following prerequisites must be met.

3.1 Universal

- Internet connectivity or at a minimum network connectivity between the backend server and clients.

3.2 Frontend

- A version of Firefox, Chrome, or IE is installed that supports javascript.

3.3 Backend

- Node.js is installed on the system

4.1 Test Cases 1: Initialize the game

4.1.1 Description

This case consists of covering all the steps required to host or join a game of checkers.

4.1.2 Pre-conditions for this test case

An internet connection, web browser running on the system, and the server is up and running

4.1.3 Scenario

Test case					
ID	Req	Description	Execution steps	Expected result	Actual result
A1	3.1.1	Open web page	1.Type the url of the checker game website in the address bar	The website loads properly and shows the option to join a game or host a game	The website loads properly and shows the option to join a game or host a game
A2	3.1.1.1	Host game - successful	1. click on the host game button	Application waits while a game ID is generated by the server; program then displays game ID	Application waits while a game ID is generated by the server; program then displays game ID
A3	3.1.1.1	Host game - too many games on server	1. click on the host game button	Application waits while server processes; server informs application too many users waiting and application informs user	Application waits while server processes; server informs application too many users waiting and application informs user

A4	3.1.1.2	Join game - successful	1. click on the join game button 2. enter game ID 3. click “ok”	A connection is established and chessboard populated with pieces	A connection is established and chessboard populated with pieces
A5	3.1.1.2	Join game - invalid game ID	1. click on the join game button 2. enter game ID 3. click “ok”	Application waits while server processes; server informs application ID is invalid and application informs user	Throws an error to the client, can’t find game they’re looking for
A6	3.1.1.2	Join game - lobby has been fill	1. click on the join game button 2. enter game ID 3. click “ok”	Application waits while server processes; server informs application lobby has been populated and application informs user	There’s a maximum person limit that says the game is full

4.2 Test Cases 2: Play the game

4.2.1 Description

This case consists of covering all the possible moves and functionalities that can be utilized while two players are engaged in a game. These cases validate whether the game logic obeys the generally accepted rules of checkers.

4.2.2 Pre-conditions for this test case

Two players need to have started a game with each other. That means that both players need to have an Internet connection, the server must be running and they have to both have successfully connected to the same game.

4.2.3 Scenario

ID	Req	Description	Execution Steps	Expected Result	Actual Result
B1	3.2	Display moves	1. Have user who's turn it is click on a game piece	Display all moves in green/red color depending on if the move jumps an opposing game piece or not	Displays all moves available to the piece in green.
B2	3.3.1	Regular forward move	1. Click on game piece that has at least one possible regular forward move (green) 2. Click on the highlighted field	Move the selected piece to the field clicked on and change turns	The piece appears on the selected field
B3	3.3.1	Regular backward move	1. Click on game piece that has at least one possible regular backward move (green) 2. Click on the	Move the selected piece to the field clicked on and change turns	The piece appears on the selected field

			highlighted field		
B4	3.3.2	Forward jump move	1. Click on game piece that has at least one possible forward jump move (red) 2. Click on the highlighted field	Move the selected piece to the field clicked on, take the piece jumped over and give that user another turn	The piece appears on the selected field and the piece jumped over is removed. If another immediate jump is available, they are then forced to move the newly moved piece again.
B5	3.3.2	Backward jump move	1. Click on game piece that has at least one possible backward jump move (red) 2. Click on the highlighted field	Move the selected piece to the field clicked on, take the piece jumped over and give that user another turn	The piece appears on the selected field and the piece jumped over is removed. If another immediate jump is available, they are then forced to move the newly moved piece again.
B6	3.2.2	Turn regular piece into king piece	1. Move a regular game piece to the furthest row	Change the look of the game piece and give it king abilities (move backwards as	The piece's visuals change according to the player that moved it and it can move in all directions

				well)	
B7	3.1.4	Check for win	1. Complete a turn	Initiate end game sequence if one player wins or there is a stalemate	The game ends

4.3 Test Case 3: Ending the game

4.3.1 Description

This case consists of all the different ways a game can end, including a player winning or someone disconnecting.

4.3.2 Pre-conditions for this test case

Two players need to have started a game with each other. That means that both players need to have an Internet connection, the server must be running and they have to both have successfully connected to the same game.

4.3.3 Scenario

ID	Req	Description	Execution Steps	Expected Result	Actual Result
C1	3.1.4	Win game	1. Take the last piece from an opponent	Display win message for the winner and loss message for the loser. Allow both users to return to homepage	Both players receive a notification of the winning player

C2	3.1.3	Stalemate	1. Reach a point with no possible moves	Display stalemate message to both users and allow them to return to homepage	The player who can't move loses.
C3	3.1.1	Quit game	1. Quit the game	Game is saved and can be re-entered at another time	Game is saved and can be re-entered at another time
C4	3.1.2	Leave game	1. Close or leave the site	Display win message to other player and allow them to return to homepage	If the game is left, then the session is saved and then the player can rejoin at a later time. Quitting doesn't end the game.
C5	3.1.5	Timeout	1. Let the game run past timeout limit	Display timeout message and return both users to the homepage after a set time	User is timed out but can rejoin the saved game

5. Appendix

5.1 Glossary

Guest/Opponent: The other player willing to join the game using the code provided by the Host player.

King: A piece becomes a “King” when it reaches the far opposite end of the board. A “King” can move diagonally forwards AND backwards.

“Take” a piece: Normally a player can move a piece only once square diagonally forward. If an opposing piece occupies that square, and the square beyond that piece isn’t occupied, then the player can move their piece an additional square, hopping over a space. This removes the opposing piece from the game. This is also known as “eating” a piece or “capturing” a piece.

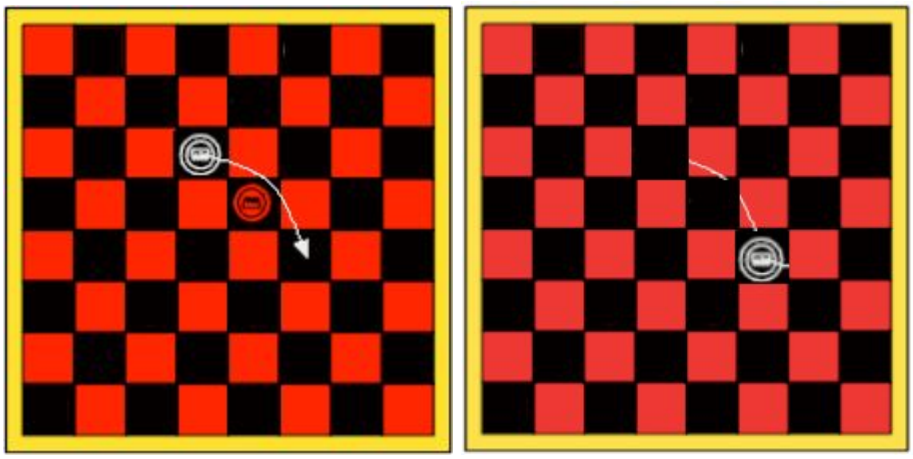

Figure 5.1

5.2 Resources

General Rules for Checkers:
http://www.indepthinfo.com/checkers/play.shtml